An Algorithm for the Solution of Dynamic Linear Programs

Mark L. Psiaki†
Cornell University, Ithaca, New York

Abstract

The algorithm's objective is to efficiently solve Dynamic Linear Programs (DLP) by taking advantage of their special staircase structure. This algorithm constitutes a stepping stone to an improved algorithm for solving Dynamic Quadratic Programs, which, in turn, would make the nonlinear programming method of Successive Quadratic Programs more practical for solving trajectory optimization problems. The ultimate goal is to bring trajectory optimization solution speeds into the realm of real-time control.

The algorithm exploits the staircase nature of the large constraint matrix of the equality-constrained DLPs encountered when solving inequality-constrained DLPs by an active set approach. A numerically-stable, staircase QL factorization of the staircase constraint matrix is carried out starting from its last rows and columns. The resulting recursion is like the time-varying Riccati equation from multi-stage LQR theory. The resulting factorization increases the efficiency of all of the typical LP solution operations over that of a dense matrix LP code. At the same time numerical stability is ensured. The algorithm also takes advantage of dynamic programming ideas about the cost-to-go by relaxing active pseudo constraints in a backwards sweeping process. This further decreases the cost per update of the LP rank-1 updating procedure, although it may result in more changes of the active set than if pseudo constraints were relaxed in a non-stagewise fashion. The usual stability of "closed-loop" Linear/Quadratic optimally-controlled systems, if it carries over to strictly linear cost functions, implies that the savings due to reduced factor update effort may outweigh the cost of an increased number of updates.

An aerospace example is presented in which a ground-to-ground rocket's distance is maximized. This example demonstrates the applicability of this class of algorithms to aerospace guidance. It also sheds light on the efficacy of the proposed pseudo constraint relaxation scheme.

Introduction

The objective of the present work is to develop and test a special-purpose algorithm for the solution of Dynamic Linear Programs (DLP). The general form of a DLP is as follows:

find: $$\mathbf{x} = \begin{bmatrix} \mathbf{x}_0^T & \mathbf{x}_1^T & \cdots & \mathbf{x}_N^T \end{bmatrix}^T \tag{1a}$$

to minimize: $$J = \begin{bmatrix} \mathbf{c}_0^T & \mathbf{c}_1^T & \cdots & \mathbf{c}_N^T \end{bmatrix} \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N \end{bmatrix} \tag{1b}$$

subject to: $$\begin{bmatrix} \mathbf{A}_{00} & \mathbf{A}_{01} & & \\ & \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{0} \\ & & \ddots & \\ \mathbf{0} & & & \mathbf{A}_{NN} \end{bmatrix} \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N \end{bmatrix} - \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_N \end{bmatrix} \left\{ \begin{matrix} = \\ \leq \end{matrix} \right\} \mathbf{0} \tag{1c}$$

where the $\mathbf{x}_i$ vectors constitute the decision vector time history, the $\mathbf{c}_i$ vectors are linear cost coefficients, and the $\mathbf{A}_{ii}$ and $\mathbf{A}_{ii+1}$ matrix blocks and the $\mathbf{b}_i$ vectors define the linear problem constraints. The bracketed equality and inequality

---

† Assistant Professor, Mechanical and Aerospace Engineering.

signs in eq. 1c indicate that both forms of constraints may be present; some rows may be equalities while others are inequalities. The problem in eq. 1a-1c is also known as a staircase LP.

A reason for interest in this problem from an aerospace controls point of view comes from its relationship to the following multi-stage trajectory optimization problem:

find: $$\mathbf{x} = \left[\mathbf{u}_0^T \; \mathbf{x}_1^T \; \mathbf{u}_1^T \; \mathbf{x}_2^T \; \ldots \; \mathbf{u}_{N-1}^T \; \mathbf{x}_N^T \right]^T \tag{2a}$$

to minimize: $$J = \sum_{k=0}^{N-1} L(\mathbf{x}_k, \mathbf{u}_k, k) \; + \phi(\mathbf{x}_N) \tag{2b}$$

subject to: $$\mathbf{x}_0 \text{ given} \tag{2c}$$

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, k) \qquad \text{for } k = 0 \ldots N-1 \tag{2d}$$

$$\mathbf{a}(\mathbf{x}_k, \mathbf{u}_k, k) \left\{ \begin{matrix} = \\ \leq \end{matrix} \right\} \mathbf{0} \qquad \text{for } k = 0 \ldots N-1 \tag{2e}$$

$$\mathbf{a}(\mathbf{x}_N, N) \left\{ \begin{matrix} = \\ \leq \end{matrix} \right\} \mathbf{0} \tag{2f}$$

where the $\mathbf{u}_i$ and $\mathbf{x}_{i+1}$ vectors constitute the control and state vector time histories, the L() and $\phi()$ functions define the nonlinear stagewise and terminal costs, eq. 2c defines the initial conditions, eq. 2d is the discrete-time dynamics difference equation, and constraints such as 2e and 2f may be present to restrict the states, or the control inputs, or both.

The current paper is part of a research program that seeks a fast and reliable way to solve the problem in eq. 2a-2f. The ultimate goal is to do real-time aerospace guidance by repeatedly solving this problem. The program is taking a two-pronged approach, algorithm improvement and parallelization of computations. This paper relates to the first prong, algorithm improvement. Fletcher's L_1 penalty function, trust region adaptation of the method of successive quadratic programs (SQP) [1] is one algorithm for solving such a nonlinear program (NP). This algorithm has fast local convergence properties, it ensures global convergence (to a local minimum), and it is good at handling inequality constraints. The application of this algorithm to the nonlinear trajectory optimization problem results in Quadratic Programming (QP) sub-problems with a special structure, the dynamic programming structure. Efficient solution of the NP requires efficient solution of the dynamic QP (DQP). Special-purpose algorithms for efficient solution of a DLP can be similar to special-purpose algorithms for efficient solution of a DQP. Thus, the present paper, in concentrating on DLP, constitutes a sort of warm-up exercise for later development of a DQP algorithm.

In addition to providing a warm-up, the present work provides a point of comparison with other research efforts in the field. Little or no work has been done on special-purpose DQP algorithms [2,3], but much attention and effort has been devoted to special-purpose DLP algorithms (e.g., Refs. 4-7). Fourer provides a useful overview of different avenues of approach that have been tried [7]. He groups algorithms for problem 1a-1c into three categories: Compact Basis, Nested Decomposition, and Transformation. All get the correct answer, but none have proved particularly successful in that none consistently out-perform the general sparse simplex method with regard to computation time.

The present algorithm is in the compact basis category; it works with a staircase factorization of the active constraints. The factorization used, a staircase QL factorization, is consistent with the plan for subsequent upgrading to handle the quadratic cost case. It has numerical stability, and there is no trade-off between numerical stability and factor compactness; general sparse matrix LP codes must deal with such trade-offs. The focus of the entire project is on aerospace guidance problems, hence the submatrices of the problem, the $\mathbf{A}_{ii}$ and $\mathbf{A}_{ii+1}$ blocks, are relatively dense. Therefore, there is hope that the current algorithm will out-perform general sparse matrix LP algorithms on these problems (e.g., algorithms such as MINOS [8]).

The body of this paper concentrates on explanation of the algorithm, with an example and conclusions at the end. Before describing the algorithm, problem 1a-1c is related to a general LP on the one hand and to a control-type LP on the other hand. The algorithm description begins with a review of the application of the L_1 penalty function method to a general LP. The staircase QL factorization then gets presented along with methods for multiplier

solution, decision vector solution, and rank-1 update. The algorithm explanation concludes with the presentation of a specialized order for problem solution that could further reduce the computational burden. The example at the end of the paper demonstrates the algorithm's applicability to aerospace trajectory optimization and examines whether the special solution ordering yields increased computational efficiency.

Equivalent Problem Forms

The problem in eq. 1a-1c is a special case of the following general LP form:

$$\text{find:} \quad \boldsymbol{\mathcal{x}} \tag{3a}$$

$$\text{to minimize:} \quad J = \mathbf{c}^T \boldsymbol{\mathcal{x}} \tag{3b}$$

$$\text{subject to:} \quad \mathbf{A}\boldsymbol{\mathcal{x}} - \mathbf{b} \left\{ \begin{matrix} = \\ \leq \end{matrix} \right\} \mathbf{0} \tag{3c}$$

where $\boldsymbol{\mathcal{x}}$ is defined in eq. 1a and $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{c}$ are defined as:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{00} & \mathbf{A}_{01} & & \\ & \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{0} \\ & & \ddots & \\ \mathbf{0} & & & \mathbf{A}_{NN} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_N \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} \mathbf{c}_0 \\ \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_N \end{bmatrix} \tag{4}$$

Form 3a-3c is fully general. It is the LP form used in developing general active constraint algorithms [1].

A more specialized DLP problem statement clarifies the relationship of these problems to controls:

$$\text{find:} \quad \mathbf{u}_k \text{ for } k = 0 \ldots N\text{-}1 \text{ and } \mathbf{x}_k \text{ for } k = 1 \ldots N \tag{5a}$$

$$\text{to minimize:} \quad J = \sum_{k=0}^{N-1} \left\{ \begin{bmatrix} \mathbf{c}_{x_k}^T & \mathbf{c}_{u_k}^T \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} \right\} + \mathbf{c}_{x_N}^T \mathbf{x}_N \tag{5b}$$

$$\text{subject to:} \quad \mathbf{x}_0 \text{ given} \tag{5c}$$

$$\mathbf{x}_{k+1} = \mathbf{F}_k \mathbf{x}_k + \mathbf{G}_k \mathbf{u}_k + \mathbf{h}_k \qquad \text{for } k = 0 \ldots N\text{-}1 \tag{5d}$$

$$\mathbf{A}_{x_k} \mathbf{x}_k + \mathbf{A}_{u_k} \mathbf{u}_k - \mathbf{b}_{xu_k} \left\{ \begin{matrix} = \\ \leq \end{matrix} \right\} \mathbf{0} \qquad \text{for } k = 0 \ldots N\text{-}1 \tag{5e}$$

$$\mathbf{A}_{x_N} \mathbf{x}_N - \mathbf{b}_{xu_N} \left\{ \begin{matrix} = \\ \leq \end{matrix} \right\} \mathbf{0} \tag{5f}$$

where there is a direct correspondence between eq. 2a-2f and eq. 5a-5f, all functions having only linear and constant terms in the latter problem. The following definitions put problem 5a-5f in the format of problem 1a-1c:

$$\boldsymbol{\mathcal{x}}_0 = \mathbf{u}_0, \quad \mathbf{A}_{00} = \begin{bmatrix} \mathbf{A}_{u_0} \\ \mathbf{G}_k \end{bmatrix}, \quad \mathbf{b}_0 = \begin{bmatrix} \mathbf{b}_{xu_0} \\ -\mathbf{h}_0 \end{bmatrix} - \begin{bmatrix} \mathbf{A}_{x_0} \\ \mathbf{F}_0 \end{bmatrix} \mathbf{x}_0, \quad \text{and } \mathbf{c}_0 = \mathbf{c}_{u_0} \tag{6a}$$

$$\boldsymbol{\mathcal{x}}_k = \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}, \quad \mathbf{A}_{kk} = \begin{bmatrix} \mathbf{A}_{x_k} & \mathbf{A}_{u_k} \\ \mathbf{F}_k & \mathbf{G}_k \end{bmatrix}, \quad \mathbf{b}_k = \begin{bmatrix} \mathbf{b}_{xu_k} \\ -\mathbf{h}_k \end{bmatrix}, \quad \text{and } \mathbf{c}_k = \begin{bmatrix} \mathbf{c}_{x_k} \\ \mathbf{c}_{u_k} \end{bmatrix} \qquad \text{for } k = 1 \ldots N\text{-}1 \tag{6b}$$

$$\mathbf{A}_{kk+1} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ -\mathbf{I} & \mathbf{0} \end{bmatrix} \qquad \text{for } k = 0 \ldots N\text{-}1 \tag{6c}$$

$$\boldsymbol{\mathcal{x}}_N = \mathbf{x}_N, \quad \mathbf{A}_{NN} = \mathbf{A}_{x_N}, \quad \mathbf{b}_N = \mathbf{b}_{x_N}, \quad \text{and } \mathbf{c}_N = \mathbf{c}_{x_N} \tag{6d}$$

Thus, the problem in eq. 1a-1c is related to controls.

Algorithm

The L_1 Exact Penalty Function and an Active Set LP Method†

The L_1 exact penalty function method enforces equality and inequality constraints as in 3c by adding a penalty cost to the problem cost that is a weighted L_1 norm of the constraint violations. The penalty function reformulation of problem 3a-3c becomes

$$\text{find:} \quad \mathbf{x} \tag{7a}$$

$$\text{to minimize:} \quad J = \mathbf{c}^T \mathbf{x} + \mu_{max}\{\|\mathbf{A}_e \mathbf{x} - \mathbf{b}_e\|_1 + \|[\mathbf{A}_{in} \mathbf{x} - \mathbf{b}_{in}]^+\|_1\} \tag{7b}$$

where μ_{max} is a large positive constant, where constraints $[\mathbf{A},\mathbf{b}]$ in eq. 3c have been split into the equality constraints, $[\mathbf{A}_e,\mathbf{b}_e]$, and the inequality constraints, $[\mathbf{A}_{in},\mathbf{b}_{in}]$, and where $[a]^+ = \max(a,0)$. This technique is actually very similar to the adjoining of constraints in Lagrange and Kuhn-Tucker formulations. The only difference is that the constraint multipliers are effectively limited in magnitude by μ_{max}. If μ_{max} is greater than the magnitude of the largest multiplier in the solution of problem 3a-3c, then problem 7a-7b has the same solution. The advantage of this technique is that it solves the problem in a single phase, optimizing while achieving feasibility. It is a variant of the LP technique known as the big M method. The primary reason for using it in the current paper is to make the algorithm compatible with the proposed method for eventually solving the NP in eq. 2a-2f.

Active constraints refer to those constraints that are satisfied as strict equalities. The active set method for solving problem 7a,7b consists of the following steps.

Main Algorithm:

1. Guess solution, $\mathbf{x}$, and split $[\mathbf{A},\mathbf{b}]$ into active and inactive rows:

$$\begin{bmatrix} \mathbf{A}_{act},\mathbf{b}_{act} \\ \mathbf{A}_{inact},\mathbf{b}_{inact} \end{bmatrix} = \mathbf{P}\,[\mathbf{A},\mathbf{b}] \tag{8}$$

   where $\mathbf{P}$ is a permutation matrix, and where the active constraints at the guessed solution must yield an $\mathbf{A}_{act}$ that is square and nonsingular.

2. Compute $\mathbf{A}_{act}^{-1}$

3. Compute $\mu_{act} = -(\mathbf{A}_{act}^{-1})^T (\mathbf{c} + \mathbf{A}^T_{inact}\mu_{inact})$ where the elements of μ_{inact} are either $-\mu_{max}$, 0, or $+\mu_{max}$ depending on whether the corresponding inactive constraint is an equality or an inequality and depending on whether it is positive or negative at the guessed solution, $\mathbf{x}$.

4. Find the element, i, of μ_{act} that is furthest out of the allowable range for the penalty function, $(\mu_{act})_i \in [-\mu_{max}, +\mu_{max}]$ for equality constraints, $(\mu_{act})_i \in [0, +\mu_{max}]$ for inequality constraints. Stop if no elements are out of range; the current $\mathbf{x}$ is optimum.

5. Compute the search direction, $\delta\mathbf{x} = \mathbf{A}_{act}^{-1}\mathbf{e}_i$, where $\mathbf{e}_i$ is the unit vector with all zeros except for a 1 in row i.

6. Compute the new guessed solution, $\mathbf{x} = \mathbf{x} + \alpha\,\delta\mathbf{x}$, where α has the same sign as $(\mu_{act})_i$ and where its magnitude is chosen to be the smallest value that makes one of the inactive constraints active, min $|\alpha|$ such that $[\mathbf{A}_{inact}(\mathbf{x} + \alpha\,\delta\mathbf{x}) - \mathbf{b}_{inact}]_j = 0$, for some row j.

---

† The discussion of this section deals with the problem form in eq. 3a-3c. The discussion is based on algorithms presented in Ref. 1.

7. Interchange rows i and j between $[\mathbf{A}_{act}, \mathbf{b}_{act}]$ and $[\mathbf{A}_{inact}, \mathbf{b}_{inact}]$, update $\mathbf{A}_{act}^{-1}$, and go to step 3.

The equation in step 3 results from differentiation with respect to $\mathbf{x}$ of the augmented cost in eq. 7b. Despite the nondifferentiability of eq. 7b for some values of $\mathbf{x}$, differentiation can be done if the active constraints are treated as adjoined equality constraints with unknown multipliers μ_{act} rather than as L_1 penalty terms. Steps 4 and 5 determine a descent direction. The step length is determined in step 6. The step length rule ensures that the entire step is in a descent direction of the piecewise-linear augmented cost. Because of the step length rule, the new active set changes by only one row. Step 7 takes advantage of this fact in the recomputation of $\mathbf{A}_{act}$'s inverse.

The algorithm used in this paper starts out by assuming that a set of pseudo constraints are active. This yields the identity matrix for the initial $\mathbf{A}_{act}$, and step 2 is trivial. The allowable range for the pseudo constraint multipliers is different than for the actual problem constraints, $\mu_{pseudo} \in [0,0]$. They get dropped from the active set in the course of the algorithm unless the problem solution is not unique.

For each constraint addition/deletion the algorithm cycles through steps 3-7. Most of the computational load per cycle is caused by manipulations with the inverse of the $\mathbf{A}_{act}$ matrix, multiplication by it in steps 3 and 5 and rank-1 updating of it in step 7. The main idea of this paper -- indeed the main idea of all compact basis schemes -- is to compactly represent factors of $\mathbf{A}_{act}$ that are suitable for carrying out multiplications by $\mathbf{A}_{act}$'s inverse and that are easy to update when $\mathbf{A}_{act}$ undergoes a rank-1 row change.

Staircase QL Factorization for Staircase LP

$\mathbf{A}_{act}$ inherits a staircase structure from A as in eq. 4. The compact QL factorization used here performs a stage-wise backwards sweep to factor the nonzero blocks of $\mathbf{A}_{act}$. The following recursion yields matrices that constitute a staircase QL factorization of $\mathbf{A}_{act}$:

$$\mathbf{D}_{NN} = \mathbf{A}_{NN_{act}} \tag{9a}$$

$$\mathbf{Q}_{kk}\begin{bmatrix} \mathbf{A}_{k\text{-}1k\text{-}1_{act}} & \mathbf{A}_{k\text{-}1k_{act}} \\ \mathbf{0} & \mathbf{D}_{kk} \end{bmatrix} = \begin{bmatrix} \mathbf{D}_{k\text{-}1k\text{-}1} & \mathbf{0} \\ \mathbf{D}_{kk\text{-}1} & \mathbf{L}_{kk} \end{bmatrix} \quad \text{for } k = N, \ldots, 1 \tag{9b}$$

$$\mathbf{Q}_{00}\mathbf{D}_{00} = \mathbf{L}_{00} \tag{9c}$$

where the $\mathbf{A}_{k\text{-}1k\text{-}1_{act}}$ and $\mathbf{A}_{k\text{-}1k_{act}}$ matrices are the nonzero blocks of the staircase $\mathbf{A}_{act}$ matrix, and where the $\mathbf{Q}_{kk}$ are orthonormal and the $\mathbf{L}_{kk}$ are lower triangular. The QL factorization is stored in the matrices $\mathbf{Q}_{kk}$, $\mathbf{L}_{kk}$, and $\mathbf{D}_{kk}$ for k = N,...,0 and the matrices $\mathbf{D}_{kk\text{-}1}$ for k = N,...,1. The $\mathbf{D}_{ij}$ matrices have no special properties. Equations 9b and 9c are only implicit relations for these factors, but the factors can be explicitly evaluated via Householder transformations. At stage k, the factorization begins with the data $\mathbf{A}_{k\text{-}1k\text{-}1_{act}}$, $\mathbf{A}_{k\text{-}1k_{act}}$, and $\mathbf{D}_{kk}$, and it computes $\mathbf{Q}_{kk}$, $\mathbf{L}_{kk}$, $\mathbf{D}_{k\text{-}1k\text{-}1}$, and $\mathbf{D}_{kk\text{-}1}$. The result $\mathbf{D}_{k\text{-}1k\text{-}1}$ then completes the necessary data for stage k-1.

Numerical stability of the factorization is ensured by the use of orthogonal transformations only. The computational complexity of the factorization algorithm is linear in the number of stages and cubic in the dimension(s) of the blocks, which is as efficient as can be expected if the blocks are dense. The factor storage is linear in the number of stages and quadratic in the dimension(s) of the blocks, which again is the best that can be achieved with dense blocks.

Equations 9a-9b are a DLP equivalent to the matrix Riccati equation of time-varying, multi-stage Linear Quadratic Regulator theory. The lower blocks of the right hand side of eq. 9b act as a closed-loop dynamic difference equation as will be shown in the next section. The upper block on the right hand side, $[\mathbf{D}_{k\text{-}1k\text{-}1}, \mathbf{0}]$, propagates active constraint effects backwards in time; it summarizes the constraints that $\mathbf{x}_{k\text{-}1}$ must satisfy in order to make possible the satisfaction of all constraints from stage k-1 onwards. Note that the number of rows in $[\mathbf{D}_{k\text{-}1k\text{-}1}, \mathbf{0}]$ does not necessarily equal the number of rows in $[\mathbf{A}_{k\text{-}1k\text{-}1_{act}}, \mathbf{A}_{k\text{-}1k_{act}}]$.

Staircase QL Solution for the Multiplier and Decision Vector Time Histories

The basic operations involved to do steps 3 and 5 of the LP algorithm presented above involve solution of a linear system by orthogonal transformation and forward or backward substitution. This is similar to the forward and backward substitutions of the simplex method and its variants. The transformations and substitutions are done in stage-wise recursions using the stagewise factors. Performing the following backward recursion then forward recursion yields the active constraint multipliers.

Backward recursion for intermediate multipliers λ_N through λ_0:

$$L_{NN}^T \lambda_N = - c_N - A_{NN_{inact}}^T \mu_{N_{inact}} - A_{N\text{-}1N_{inact}}^T \mu_{N\text{-}1_{inact}} \tag{10a}$$

$$L_{kk}^T \lambda_k = - D_{k+1k}^T \lambda_{k+1} - c_k - A_{kk_{inact}}^T \mu_{k_{inact}} - A_{k\text{-}1k_{inact}}^T \mu_{k\text{-}1_{inact}} \qquad \text{for } k = N\text{-}1, \ldots, 1 \tag{10b}$$

$$L_{00}^T \lambda_0 = - D_{10}^T \lambda_1 - c_0 - A_{00_{inact}}^T \mu_{0_{inact}} \tag{10c}$$

Forward recursion for intermediate multipliers β_0 through β_N and for active constraint multipliers $\mu_{0_{act}}$ through $\mu_{N_{act}}$:

$$\beta_0 = Q_{00}^T \lambda_0 \tag{11a}$$

$$\begin{bmatrix} \mu_{k_{act}} \\ \beta_{k+1} \end{bmatrix} = Q_{k+1k+1}^T \begin{bmatrix} \beta_k \\ \lambda_{k+1} \end{bmatrix} \qquad \text{for } k = 0 \ldots N\text{-}1 \tag{11b}$$

$$\mu_{N_{act}} = \beta_N \tag{11c}$$

Step 5 of the LP algorithm is accomplished by solving the system of equations $A_{act}\delta x = e_i$. To illustrate how the staircase QL factorization does this, the following equations present its use in solving the alternate system $A_{act}x = b_{act}$. Again, a backward recursion followed by a forward recursion yields the solution.

Backward recursion for intermediate nonhomogeneous constraint terms d_N through d_0 and g_N through g_0:

$$d_N = b_{N_{act}} \tag{12a}$$

$$\begin{bmatrix} d_{k-1} \\ g_k \end{bmatrix} = Q_{kk} \begin{bmatrix} b_{k-1_{act}} \\ d_k \end{bmatrix} \qquad \text{for } k = N, \ldots, 1 \tag{12b}$$

$$g_0 = Q_{00}\, d_0 \tag{12c}$$

Forward recursion for the decision vectors x_0 through x_N:

$$L_0 x_0 = g_0 \tag{13a}$$

$$L_{kk} x_k = - D_{kk-1} x_{k-1} + g_k \qquad \text{for } k = 1 \ldots N \tag{13b}$$

As stated earlier, eq. 13b is like the closed-loop dynamic difference equation of multi-stage LQR theory. The matrix L_{kk} is lower triangular and allows for easy solution for x_k in terms of x_{k-1} and g_k.

Rank-1 Update of Staircase QL Factorization

This section explains how to efficiently update the staircase QL factorization after a single constraint addition/deletion. This procedure must be carried out every time step 7 of the main algorithm is encountered. One could recompute the entire factorization, but the practicality of all LP codes hinges on their ability to update the factors for much less work than would be required to recompute them from scratch.

The general add/drop updating scheme for the staircase QL factorization must update the results of eq. 9a-9b when an arbitrary row j at stage kadd gets added to the active constraint set and another arbitrary row i at stage kdrop gets deleted from the active constraint set. Thus, $[A_{kk_{act}}, A_{kk+1_{act}}]|_{k=kadd}$ gets a new row and $[A_{kk_{act}}, A_{kk+1_{act}}]|_{k=kdrop}$ loses a row. The stages kadd and kdrop can have any relationship to each other, and the update algorithm must be able to handle all possible cases. Three different cases can occur, kadd > kdrop, kadd = kdrop, and kadd < kdrop.

Efficient rank-1 update can be accomplished by a series of stage-wise rank-1 updates linked together in an appropriate manner. Three different stagewise rank-1 updating algorithms are needed to do this. The first algorithm updates the factors computed in eq. 9b when a new row has been added to the bracketed expression on the left-hand side of that equation. The second algorithm updates these same factors in the case of a row deletion from the bracketed expression on the left-hand side. This second algorithm also modifies $Q_{k\text{-}1k\text{-}1}$ by a single Householder transformation. The third stagewise algorithm updates these same factors when D_{kk} has undergone an arbitrary rank-1 change. Recall that the bracketed matrix on the left-hand side of eq. 9b represents the input data for a given stage and the Q_{kk} matrix together with the bracketed expression on the right-hand side represents the result of the stagewise factorization. The following discussion explains each of these three algorithms and the way in which they work together to accomplish the multi-stage rank-1 update.

First, consider what happens to the stage k factorization when a new row gets added to either $[A_{k\text{-}1k\text{-}1_{act}}, A_{k\text{-}1k_{act}}]$ or D_{kk}. The algorithm begins by adding a row and a column to Q_{kk} with all 0s except for a 1 at the intersection of the new row and the new column. Thus, Q_{kk} remains orthonormal. Suppose the new constraint row is $[p^T_{k1}, p^T_{k2}]$, then the new row and column of Q_{kk} are added so that eq. 9b temporarily becomes:

$$\begin{bmatrix} Q_{kk_{11}} & 0 & Q_{kk_{12}} \\ 0^T & 1 & 0^T \\ Q_{kk_{21}} & 0 & Q_{kk_{22}} \end{bmatrix} \begin{bmatrix} A_{k\text{-}1k\text{-}1_{act}} & A_{k\text{-}1k_{act}} \\ p^T_{k1} & p^T_{k2} \\ 0 & D_{kk} \end{bmatrix} = \begin{bmatrix} D_{k\text{-}1k\text{-}1} & 0 \\ p^T_{k1} & p^T_{k2} \\ D_{kk\text{-}1} & L_{kk} \end{bmatrix} \tag{14}$$

where the $Q_{kk_{ij}}$ matrix blocks are just the blocks of the original Q_{kk} matrix. Suppose n_k is the dimension of the x_k decision vector. Then it is also the dimension of the square lower-triangular matrix L_{kk}. A series of n_k Givens rotations can be performed to zero out p^T_{k2} while preserving the lower-triangular structure of L_{kk}. The first Givens rotation uses the last row of L_{kk} as the pivot row and zeros out the last element of p^T_{k2}, and successive rotations use successively higher rows of L_{kk} as the pivot and zero out successive elements of p^T_{k2} going from right to left. Suppose these rotations are G_1 to G_{n_k}. Then the new stage k factorization becomes:

$$Q_{kk_{new}} = G_{n_k} \cdot \ldots \cdot G_1 \cdot \begin{bmatrix} Q_{kk_{11}} & 0 & Q_{kk_{12}} \\ 0^T & 1 & 0^T \\ Q_{kk_{21}} & 0 & Q_{kk_{22}} \end{bmatrix} \tag{15a}$$

$$\begin{bmatrix} D_{k\text{-}1k\text{-}1} & 0 \\ d^T_{k1} & 0^T \\ D_{kk\text{-}1_{new}} & L_{kk_{new}} \end{bmatrix} = G_{n_k} \cdot \ldots \cdot G_1 \cdot \begin{bmatrix} D_{k\text{-}1k\text{-}1} & 0 \\ p^T_{k1} & p^T_{k2} \\ D_{kk\text{-}1} & L_{kk} \end{bmatrix} \tag{15b}$$

$$D_{k\text{-}1k\text{-}1_{new}} = \begin{bmatrix} D_{k\text{-}1k\text{-}1} \\ d^T_{k1} \end{bmatrix} \tag{15c}$$

where the last equation has been included to emphasize the fact that the new $D_{k\text{-}1k\text{-}1}$ differs from the old $D_{k\text{-}1k\text{-}1}$ by only a single new row. This fact sets the stage for the use of this same algorithm at stage k-1. The new Q_{kk} is orthonormal because the augmented matrix is orthonormal and because all Givens rotations are orthonormal. Thus, the new factors have all of the required properties for use in the LP algorithm described above.

Next, consider what happens to the stage k factorization when a row gets deleted from either $[A_{k\text{-}1k\text{-}1_{act}}, A_{k\text{-}1k_{act}}]$ or D_{kk}. The following development is based on ideas for QP from Ref. 9. Write Q_{kk} in the form

$$Q_{kk} = \begin{bmatrix} Q_{kk11} & q_{kk12} & Q_{kk13} \\ q^T_{kk21} & q_{kk22} & q^T_{kk23} \\ Q_{kk31} & q_{kk32} & Q_{kk33} \end{bmatrix} \tag{16}$$

where the middle column conforms in matrix multiplication with the constraint row that is getting deleted -- rows of D_{kk} can be referred to as constraints; they are propagated active constraints. The bottom blocks, $[Q_{kk31}, q_{kk32}, Q_{kk33}]$, have n_k rows, the same as in the bottom blocks on the right hand side of eq. 9b, $[D_{kk-1}, L_{kk}]$.

The stagewise deletion algorithm starts with a Householder transformation in which the q_{kk22} row in the above representation is used as the pivot row to zero out q_{kk12} in the first rows. Next, a series of n_k Givens rotations is used to zero out successive elements of q_{kk32} starting with the topmost element and working downwards. Again, the q_{kk22} row in the above representation is used as the pivot. If the Householder transformation is **H** and the Givens rotations are G_1 to G_{n_k}, then the following changes to the stage k factorization result:

$$\begin{bmatrix} Q_{kk11_{new}} & 0 & Q_{kk12_{new}} \\ 0^T & 1 & 0^T \\ Q_{kk21_{new}} & 0 & Q_{kk22_{new}} \end{bmatrix} = G_{n_k} \bullet \ldots \bullet G_1 \bullet H \bullet \begin{bmatrix} Q_{kk11} & q_{kk12} & Q_{kk13} \\ q^T_{kk21} & q_{kk22} & q^T_{kk23} \\ Q_{kk31} & q_{kk32} & Q_{kk33} \end{bmatrix} \tag{17a}$$

$$Q_{kk_{new}} = \begin{bmatrix} Q_{kk11_{new}} & Q_{kk12_{new}} \\ Q_{kk21_{new}} & Q_{kk22_{new}} \end{bmatrix} \tag{17b}$$

$$\begin{bmatrix} D_{k\text{-}1k\text{-}1_{new}} & 0 \\ p^T_{k1} & p^T_{k2} \\ D_{kk\text{-}1_{new}} & L_{kk_{new}} \end{bmatrix} = G_{n_k} \bullet \ldots \bullet G_1 \bullet H \bullet \begin{bmatrix} D_{k\text{-}1k\text{-}1} & 0 \\ D_{kk\text{-}1} & L_{kk} \end{bmatrix} \tag{17c}$$

where $[p^T_{k1}, p^T_{k2}]$ corresponds to the constraint that is getting dropped. Orthonormality of the original Q_{kk} matrix ensures the form of the result on the left-hand side of eq. 17a. Note that the matrix $D_{k\text{-}1k\text{-}1_{new}}$ is a function only of $D_{k\text{-}1k\text{-}1}$ and **H**; the Givens rotations do not affect it. Therefore, another Householder transformation, **H"**, can be constructed based on the same Householder vector. It yields:

$$\begin{bmatrix} D_{k\text{-}1k\text{-}1_{new}} \\ d^T_{k\text{-}1_{drop}} \end{bmatrix} = H'' \, D_{k\text{-}1k\text{-}1} \tag{18}$$

where $d_{k\text{-}1_{drop}}$ is not necessarily equal to p_{k1}. This sets the stage for propagation of the constraint deletion process backwards to stage k-1. If $Q_{k\text{-}1k\text{-}1}$ gets transformed according to

$$Q_{k\text{-}1k\text{-}1_{interim}} = Q_{k\text{-}1k\text{-}1} \begin{bmatrix} I & 0 \\ 0 & H'' \end{bmatrix} \tag{19}$$

then $Q_{k\text{-}1k\text{-}1_{interim}}$ is still orthonormal because **H"** is orthonormal, and because **H"** is equal to its transpose, constraint $[0^T, d^T_{k\text{-}1_{drop}}]$ is the constraint that must get dropped at stage k-1. The foregoing algorithm can accomplish the deletion at this next preceding stage.

The algorithm that performs the multi-stage rank-1 update of the staircase QL factorization starts with the highest stage at which either a constraint addition or deletion occurs. It uses whichever of the two foregoing stagewise updating algorithms is appropriate to propagate the addition or deletion backwards. It continues until it reaches a stage at which both an addition and a deletion must take place. One, but not both, of the changes at this stage may be the result of a backwards propagation. At this stage of the concurrent add/drop, the multi-stage algorithm first does a single-stage constraint addition followed by a single-stage constraint deletion with no change of stage in between.

If the index of this stage is k, then $\mathbf{D}_{k\text{-}1k\text{-}1}$ will differ from its pre-update value by a rank-1 change at most. This can be shown by recognizing that the result on the left-hand side of eq. 15c becomes the input data on the right-hand side of eq. 18 when an add followed by a drop both occur at the same stage:

$$\begin{bmatrix} \mathbf{D}_{k\text{-}1k\text{-}1_{new}} \\ \mathbf{d}^T_{k\text{-}1_{drop}} \end{bmatrix} = \mathbf{H}'' \begin{bmatrix} \mathbf{D}_{k\text{-}1k\text{-}1} \\ \mathbf{d}^T_{k1} \end{bmatrix} \tag{20}$$

$\mathbf{H}''$ is a Householder transformation; it differs from the identity matrix by a rank-1 matrix, hence the conclusion about the change in $\mathbf{D}_{k\text{-}1k\text{-}1}$. Define this rank-1 change in terms of the vectors $\mathbf{r}_{k\text{-}1}$ and $\mathbf{s}_{k\text{-}1}$:

$$\mathbf{D}_{k\text{-}1k\text{-}1_{new}} = \mathbf{D}_{k\text{-}1k\text{-}1} + \mathbf{r}_{k\text{-}1}\mathbf{s}^T_{k\text{-}1} \tag{21}$$

If either $\mathbf{r}_{k\text{-}1}$ or $\mathbf{s}_{k\text{-}1}$ is the $\mathbf{0}$ vector, then the multi-stage rank-1 update is complete. If not, then another stagewise updating algorithm is needed.

The final stagewise updating algorithm must update the stagewise factors for an arbitrary rank-1 change in the data $\mathbf{D}_{kk}$. It is allowed to produce at most a rank-1 change in $\mathbf{D}_{k\text{-}1k\text{-}1}$. This restriction on its effect on $\mathbf{D}_{k\text{-}1k\text{-}1}$ makes it self recursive for all subsequent stagewise factorizations in the backwards chain. It can be used for updating the factorizations of all stages that precede the concurrent constraint addition/deletion stage. It can be used recursively until no more updating is needed.

One might suppose that the necessary algorithm has already been developed in a work such as Ref. 10. That paper is a good reference for rank-1 modifications, and it defines the general methodology used in the algorithm below, but the relevant algorithm from [10] would result in a rank-2 change to $\mathbf{D}_{k\text{-}1k\text{-}1}$. This would destroy the stagewise recursive applicability of the algorithm, hence the modified algorithm presented below.

Suppose there has been a rank-1 modification to $\mathbf{D}_{kk}$ as in eq. 21 (except at stage k instead of stage k-1). Then, eq. 9b gets modified:

$$\mathbf{Q}_{kk}\begin{bmatrix} \mathbf{A}_{k\text{-}1k\text{-}1_{act}} & \mathbf{A}_{k\text{-}1k_{act}} \\ \mathbf{0} & \left[\mathbf{D}_{kk}+\mathbf{r}_k\mathbf{s}^T_k\right] \end{bmatrix} = \begin{bmatrix} \mathbf{D}_{k\text{-}1k\text{-}1} & \mathbf{0} \\ \mathbf{D}_{kk\text{-}1} & \mathbf{L}_{kk} \end{bmatrix} + \begin{bmatrix} \mathbf{v}_{k\text{-}1} \\ \mathbf{w}_k \end{bmatrix}\begin{bmatrix} \mathbf{0}^T & \mathbf{s}^T_k \end{bmatrix} \tag{22}$$

where

$$\begin{bmatrix} \mathbf{v}_{k\text{-}1} \\ \mathbf{w}_k \end{bmatrix} = \mathbf{Q}_{kk}\begin{bmatrix} \mathbf{0} \\ \mathbf{r}_k \end{bmatrix} \tag{23}$$

and where $\mathbf{v}_{k\text{-}1}$ and $\mathbf{D}_{k\text{-}1k\text{-}1}$ have the same number of rows, $n_{dk\text{-}1}$. The algorithm starts by reducing the v-w vector to a vector with zeros in all of its entries except the last two. This is done by first applying a Householder transformation, $\mathbf{H}_1$, to the first $n_{dk\text{-}1}+1$ rows to zero out the first $n_{dk\text{-}1}$ rows. Then a series of Givens rotations, $\mathbf{G}_1$ to $\mathbf{G}_{nk\text{-}1}$, is applied to successive pairs of rows of the resulting vector to zero out successive elements until only the last two elements are left nonzero. These same transformations are applied to all terms on both sides of eq. 22, and the two terms on the right hand side of the equation are added together with the following (partial) result:

$$G_{n_k-1}\bullet\ldots\bullet G_1\bullet H_1\bullet\left\{\begin{bmatrix}0\\ L_{kk}\end{bmatrix}+\begin{bmatrix}v_{k-1}\\ w_k\end{bmatrix}s^T{}_k\right\}=\begin{bmatrix}\alpha v_{k-1} & 0 & 0 & \ldots & 0\\ * & * & & & \\ * & * & * & & 0\\ * & * & * & * & \\ \vdots & \vdots & \vdots & \vdots & \ddots\\ * & * & * & * & *\\ * & * & * & * & *\end{bmatrix} \tag{24}$$

where α is a scalar and where asterisks (*) indicate nonzero scalar elements of the matrix. The top row of vector entries in the right hand matrix corresponds to the upper right-hand **0** block in the bracketed expression on the right side of eq. 9b. The bottom rows constitute a matrix with nonzero elements on the first diagonal above the main, on the main diagonal, and below the main diagonal. All elements on diagonals that are 2 or more above the main diagonal are zero. The nonzero entry in the first column of the top block, αv_{k-1}, results from application of the $\mathbf{H}_1$ Householder transformation to matrix $[\mathbf{0}^T,\mathbf{L}^T{}_{kk}]^T$.

The remaining transformations are applied in order to restore lower triangularity to the matrix on the right hand side of eq. 24. First, a series of n_k-1 Givens rotations, $\mathbf{G}_{n_k}$ to $\mathbf{G}_{2n_k-2}$, is applied to successive pairs of rows of the matrix starting from the last two rows and working up to the first two rows in the lower block. Each rotation zeros out one of the above-diagonal elements. At the end of this operation the matrix has the form

$$\begin{bmatrix}\alpha v_{k-1} & 0 & 0 & \ldots & 0\\ * & & & & \\ * & * & & & 0\\ * & * & * & & \\ \vdots & \vdots & \vdots & \ddots & \\ * & * & * & * & \\ * & * & * & * & *\end{bmatrix} \tag{25}$$

so that the lower block is lower triangular. The final part of the algorithm is to apply a last Householder transformation, $\mathbf{H}_2$, to zero out the first column of the top block. These operations result in the following factor updates:

$$\mathbf{Q}_{kk_{new}}=\mathbf{H}_2\bullet\mathbf{G}_{2n_k-2}\bullet\ldots\bullet\mathbf{G}_1\bullet\mathbf{H}_1\bullet\mathbf{Q}_{kk} \tag{26a}$$

$$\begin{bmatrix}\mathbf{D}_{k-1k-1_{new}} & 0\\ \mathbf{D}_{kk-1_{new}} & \mathbf{L}_{kk_{new}}\end{bmatrix}=\mathbf{H}_2\bullet\mathbf{G}_{2n_k-2}\bullet\ldots\bullet\mathbf{G}_1\bullet\mathbf{H}_1\bullet\left\{\begin{bmatrix}\mathbf{D}_{k-1k-1} & 0\\ \mathbf{D}_{kk-1} & \mathbf{L}_{kk}\end{bmatrix}+\begin{bmatrix}v_{k-1}\\ w_k\end{bmatrix}\begin{bmatrix}0^T & s^T{}_k\end{bmatrix}\right\} \tag{26b}$$

In both of the Householder transformations, the first $n_{d_{k-1}}$ elements of the transformation vector are parallel to v_{k-1}, and none of the Givens rotations affect the first $n_{d_{k-1}}$ rows of the bracketed expression on the right of eq. 26b. Therefore, $\mathbf{D}_{k-1k-1_{new}}$ differs from $\mathbf{D}_{k-1k-1}$ only by a rank-1 matrix:

$$\mathbf{D}_{k-1k-1_{new}}=\mathbf{D}_{k-1k-1}+v_{k-1}y^T{}_{k-1} \tag{27}$$

where the vector y_{k-1} can be determined from the algorithm presented above. Thus, the algorithm updates stage k according to the rank-1 change in the stage's input data, and it produces a similar rank-1 change in the input data for stage k-1. The multi-stage rank-1 updating algorithm propagates these rank-1 changes backwards until at some stage one or both of the vectors in the rank-1 change are zero. This occurs at least by the time stage k = 0 is reached because $\mathbf{D}_{-1-1}$ has zero dimension.

Numerical stability of the factorization update is ensured by the use of orthogonal transformations only. The computational complexity of the multi-stage update algorithm is linear in the number of stages affected and quadratic in the dimension(s) of the blocks, which is as efficient as can be expected if the blocks are dense. If the number of stages affected by a particular row interchange of active and inactive constraints can kept small, then the cost of the update will be small. This fact provides the motivation for the solution scheme presented in a later section.

Possible Improvements to Banded Staircase QL Factorization

Several issues come to mind in considering the forgoing use of a QL factorization for an LP basis factorization. They all revolve around a single question: is the entire factorization needed to implement the LP algorithm? For instance, a general LP method has been developed that uses LQ factorization but does not store Q [11]. Not storing the Q factors would yield a great savings in memory and computation time if it carried over to the present multi-stage algorithm. This presents no difficulty to the procedures for solving for the multiplier and decision vectors, steps 3 and 5 of the main LP algorithm. The problem with not storing Q occurs in the factor update, step 7. There is no apparent way to do the single-stage constraint deletion or the single-stage rank-1 modification without storing at least some of the $\mathbf{Q_{kk}}$ matrix. Reference 9 has some ideas in its section on quadratic programming that could be used to eliminate storage of the lower part of $\mathbf{Q_{kk}}$. Alternatively, storage of $\mathbf{D_{kk}}$ and $\mathbf{D_{kk-1}}$ could be eliminated. Savings in computation time and memory would be about the same for either scheme, about 30% savings. These issues may be explored in a later work.

Backwards-Sweeping Pseudo Constraint Relaxation and an Alternate Method of Selecting the Active Constraint to Drop

In theory, all dynamic programming problems can be solved by first computing the cost-to-go at each stage, then solving a single stage optimization at each stage. Part of the cost for each of these single stage problems is the cost-to-go that results from the stage's decisions. For DLPs and for their associated L_1 penalty function problems, the cost-to-go at a given stage is a piecewise-linear convex function of the decisions at that stage. This convexity property gives rise to a hope that DLPs may have a property like the stability property of their quadratic-cost counterparts, multi-stage LQR problems. In the DLP context, this property might mean that a small change in the decisions at a given stage would give rise to even smaller changes in the state at subsequent stages. This might translate into a grouping of constraint additions and deletions at stages nearly following the stage at which the decision variations are taking place.

If this property exists, it can be exploited without the necessity of computing the entire cost-to-go function. If all of the active constraint multipliers for constraints following a given stage are within their allowable range, then the guessed solution is an optimal trajectory for all stages following that stage. Also, the local linear piece of the cost-to-go function is known. Suppose the given stage can be optimized without causing any of the multipliers at subsequent stages to exceed their L_1 penalty function bounds. Suppose also that all of the original pseudo constraints are active for the preceding stages. Then, the rank-1 updates that would have to be done during the optimization of that stage might involve changes to very few stages. The assumption about the the pseudo constraints ensures that the updates will not affect any of the stages preceding stage k-1 if stage k is being optimized. The possibility of stability implies that max(kadd,kdrop) might, in most cases, not be much larger than k.

A change is needed to the main LP procedure presented above. It allows the multipliers at stages subsequent to the stage being optimized to vary outside of their L_1 penalty function bounds. The modification needs to be in the selection of the active constraint that gets dropped on each cycle. In the main algorithm, the dropped constraint is the same as the non-optimal constraint that gets relaxed in steps 4-6. This could cause an active constraint multiplier that was within its bounds to go out of its bounds. If the multiplier corresponded to a constraint at a subsequent stage, then the optimality of the subsequent stages would break down.

This situation can be avoided by performing a search in the active constraint multiplier space for the active constraint to be dropped. This search is the dual of that carried out in steps 5 and 6 of the main algorithm, and the search direction is defined by relaxing the L_1 penalty function constraint on the multiplier associated with inactive constraint j, the inactive constraint that is becoming active. The size of the step in multiplier space is chosen to be the smallest that brings one of the active constraint multipliers to a bound which the multiplier would violate if the step size were larger; the new active constraint must be included in this test. The active constraint whose multiplier

bound limits this step size is the active constraint that gets dropped. It is not necessarily the constraint whose relaxation was dictated in step 3 of the main algorithm.

With this scheme in place, only pseudo constraints will have multipliers that are out of bounds in the step 3 optimality test. The number of non-optimal active constraints will never increase. In turn, each pseudo constraint will eventually be the constraint that gets chosen for dropping, though this may happen while another pseudo constraint is being relaxed.

A special order has been chosen for relaxing pseudo constraints to take advantage of the possibility of savings from "stability". The modified main algorithm starts by testing and relaxing only stage-N pseudo constraints in steps 3-6. This continues until all of the stage-N pseudo constraints have dropped from the active list or have zero multipliers. Then the algorithm switches to exclusive consideration of the stage-(N-1) pseudo constraints in steps 3-6. It performs add/drop cycles until all of these pseudo constraints get dropped or have zero multipliers. It continues this stagewise pseudo constraint relaxation scheme in a backwards sweep all the way to stage 0. The guessed solution is optimal after the last stage-0 pseudo constraint has been dropped or has had its multiplier go to zero. The trajectory from stage k to stage N is an optimal trajectory once all of the stage-k pseudo constraints have been dropped or have had their multipliers go to zero (although it probably will not be the final optimal trajectory associated with the solution to the overall problem).

Comparison of Algorithm Complexity with Matrix Riccati Equation

Table 1 compares the present algorithm's computational complexity with that of related algorithms for a typical aerospace controls problem. The time-varying multi-stage Matrix Riccati equation actually does not compute an A_{act} because it solves a different optimization problem. It has been included because control engineers are more familiar with it. The three QL factorization entries assume that the factors are built up from initial pseudo constraints via 900 rank-1 updates. In the last two entries, assumptions are made about the average number of stages affected per rank-1 update. The table clearly indicates that the staircase QL factorization makes a tremendous improvement in comparison to the dense factorization; the improvement will not be nearly so great in comparison to a general sparse matrix code. Also, large improvements are expected from the special ordering of the pseudo constraint relaxation. Note that all of the algorithms are far more costly than the implementation of a time-varying LQR solution. Inequality constraints are difficult to handle.

Table 1.

A Comparison of Effort for Factorization of A_{act} for a Typical Aerospace Control Example
(100 stages, 6 state vector elements, 3 control vector elements)

Solution Method	Effort (No. of Mult., Div., & Sqrt.)
Matrix Riccati Equation	112,000
Dense Matrix QL, Not storing Q	2,920,000,000
Staircase QL, Arbitrary order of pseudo constraint relaxation	90,700,000
Staircase QL, Special order of pseudo constraint relaxation	4,400,000

Aerospace Example

A simple aerospace control problem has been solved with the algorithm in order to demonstrate the usefulness of this class of algorithms on aerospace problems and in order to study the algorithm's behavior. The problem is one of fixed-time maximization of the distance travelled by a thrust- and impulse-limited ground-to-ground rocket. The continuous-time problem is:

find: $\mathbf{u}(t)$ for $0 \le t \le t_f = 12$ sec (28a)

to minimize: $J = -[1\ 0\ 0\ 0]\mathbf{x}(t_f)$ (28b)

subject to: $\mathbf{x}(0) = \mathbf{0}$ (28c)

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 & 0 \\ 32.2 & 0 \\ 0 & 0 \\ 0 & 32.2 \end{bmatrix} u + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -32.2 \end{bmatrix} \tag{28d}$$

$$\|u(t)\|_2 \leq 5 \text{ g} \tag{28e}$$

$$\int_0^t \|u(\tau)\|_2 d\tau \leq 10 \text{ g-sec} \tag{28f}$$

$$-[0\ 0\ 1\ 0]x(t) \leq 0 \text{ ft.} \tag{28h}$$

which is a point-mass model of motion in the vertical plane. The acceleration (thrust) limit is 5 gs and the impulse limit is 10 g-sec. The first two state vector elements are horizontal position and velocity; the last two elements are vertical position and velocity. Only thrust and a uniform gravity field act on the rocket. The first control vector element is horizontal acceleration; the second element is vertical acceleration. Constraint 28h keeps the rocket above the ground.

In order solve this problem with this paper's algorithm, the control time history has been approximated by a 24-stage zero-order hold. Additionally, the norms in constraints 28e and 28f have been approximated by functions with octagonally-shaped contours. The 2-norm's contours are spherical; so, this approximation introduces some modelling error. Fixing the end time seems unnatural, but it is necessary in order to be able to model the problem as an LP. An NP model is needed to handle the free-end-time case.

The LP code solved this problem in 55 min. on an IBM PC-AT with an 80287 coprocessor. It started from a first guess that violated inequality constraint 28h at every stage and that foolishly tried to maintain a constant thrust for the entire trajectory. Figures 1-3 compare the multi-stage LP solution with the exact continuous-time solution. In Fig. 1, the LP solution does better than the exact solution because of mis-modeling; it takes advantage of some extra thrust available at some points of the octagon norm. The thrust magnitude and angle time histories, Fig. 2 & 3, are both close to the exact solution, and the discrepancies are due to the same modeling error.

Figure 4 gives a 2-dimensional histogram of the constraint addition/deletion frequency. The left-hand horizontal axis indicates the stage at which the pseudo constraints are being relaxed in the special backwards-chaining process. The right-hand horizontal axis indicates the stage at which constraint additions and deletions are occurring during that relaxation process. The vertical axis gives the frequency of additions/deletions at the given right-hand-axis stage during pseudo-constraint relaxation at the given left-hand-axis stage. The extreme left-hand side of the figure shows no constraint addition or deletions -- none can occur at any stage before stage k-1 when the pseudo constraints at stage k are the ones being relaxed. The peaks on and near the center diagonal of the graph lend support to the conjecture that most of the constraint additions/deletions will happen at stages near the pseudo-constraint-relaxation stage. Note, however, that a moderate amount of constraint addition/deletion activity occurred near the terminal stage throughout the optimization. Nevertheless, the average factor update was relatively cheap. Altogether, about 800 rank-1 updates occur during the optimization. The total number of decision vectors in the time history is 312 -- 9 extra states are needed to model the impulse constraint in eq. 28f.

Conclusions

An algorithm has been presented for solving Dynamic Linear Programs. It takes advantage of the staircase structure of the active constraint matrix by factorizing it into staircase QL factors. These are derived in a stagewise fashion and play a role similar to that played by the time-varying matrix Riccati equation in multi-stage LQR theory. All of the usual linear programming functions have been implemented with the staircase QL factorization: decision vector solution, multiplier solution, and rank-1 updating. Each function has a computational complexity of $O(n^2N)$ or less, where n is a block dimension and N is the number of stages. This is the best that can be expected for dense blocks. Numerical stability is assured via the exclusive use of QL factors and is independent of pivoting strategies.

The algorithm is a modified active set implementation of the big M method with pseudo-constraint initialization. The modification restricts the set of non-optimal constraints that can be relaxed at one time to a single stage. This restriction gets iterated through all the stages in a backwards chain. Also, the modification chooses the

constraint that gets dropped in a way that assures optimality of the final portion of the solution time history. The modified strategy's goal is to reduce the average complexity of the rank-1 updates.

A 24-stage example problem has been solved. The algorithm solves the 312-dimensional problem in about 800 add/drop cycles, requiring 55 min. on an IBM PC-AT. The average update complexity is significantly reduced by the modified active set strategy.

Acknowledgement

This research was supported in part by the National Aeronautics and Space Administration under Grant No. NAG-1-1009.

References

1. Fletcher, R., **Practical Methods of Optimization**, 2nd Edition, J. Wiley & Sons, (New York 1987).
2. Dantzig, G.B., Personal Communication, Aug. 1989.
3. Fourer, R., Personal Communication, Aug. 1989.
4. Dantzig. G.B., "Programming of Interdependent Activities II: Mathematical Model," **Econometrica**, Vol. 17, 1949, pp. 200-211.
5. Dantzig, G.B., and Wolfe, P., "Decomposition Principle for Linear Programs," **Operations Research**, Vol. 8, Jan.-Feb. 1960, pp. 101-111.
6. Propoi, A., and Krivonozhko, V., "The Simplex Method for Dynamic Linear Programs," Proceedings of the IIASA Workshop on Large-Scale Linear Programming, June 2-6, 1980, (Laxenburg, Austria, 1981), pp. 299-363.
7. Fourer, R., "Solving Staircase Linear Programs by the Simplex Method," Proceedings of the IIASA Workshop on Large-Scale Linear Programming, June 2-6, 1980, (Laxenburg, Austria, 1981), pp. 179-259.
8. Murtaugh, B.A., and Saunders, M.A., "MINOS 5.0 Users Guide," Report SOL 83-20, Department of Operations Research, Stanford U., (Stanford, California, 1983).
9. Coleman, T.F., **Large Sparse Numerical Optimization**, Springer-Verlag, (New York, 1984)
10. Gill, P.E., Golub, G.H., Murray, W., and Saunders, M.A., "Methods for Modifying Matrix Factorizations," **Mathematics of Computation**, Vol. 28, April 1974, pp. 505-535.
11. Saunders, M., "Large-Scale Linear Programming Using the Cholesky Factorization," Ph.D. Dissertation, Stanford University, (Stanford, California, 1972).

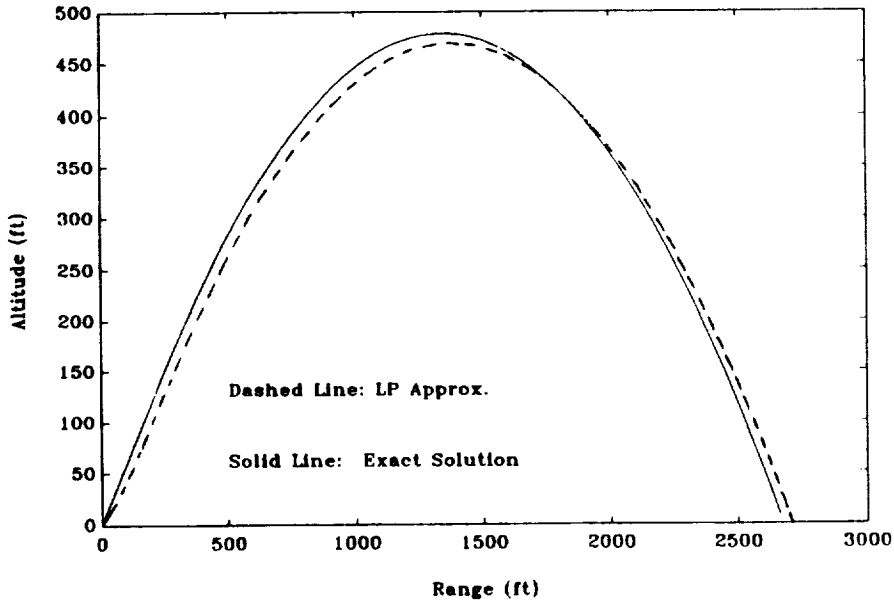

Fig. 1 Altitude vs. Range Trajectory for Ground-to-Ground Missile Range Maximization

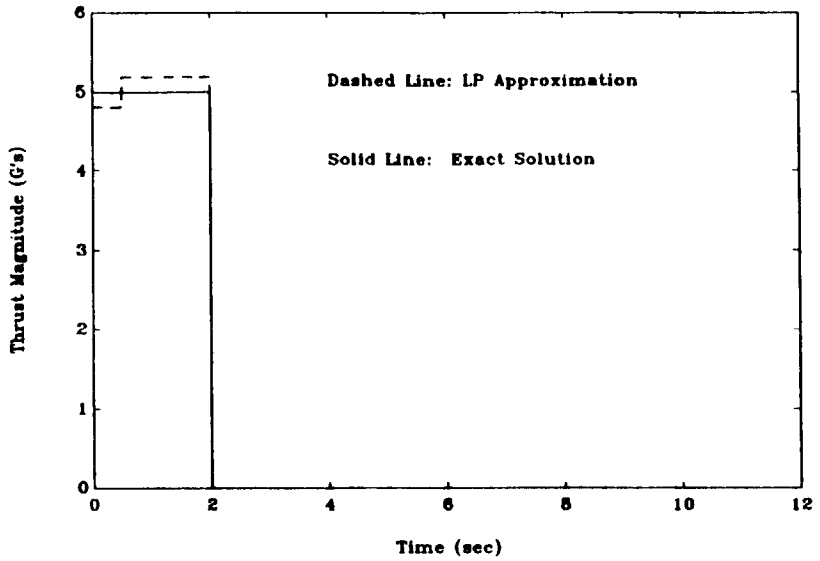

Fig. 2 Thrust (Acceleration) Magnitude Time History for Ground-to-Ground Missile Range Maximization

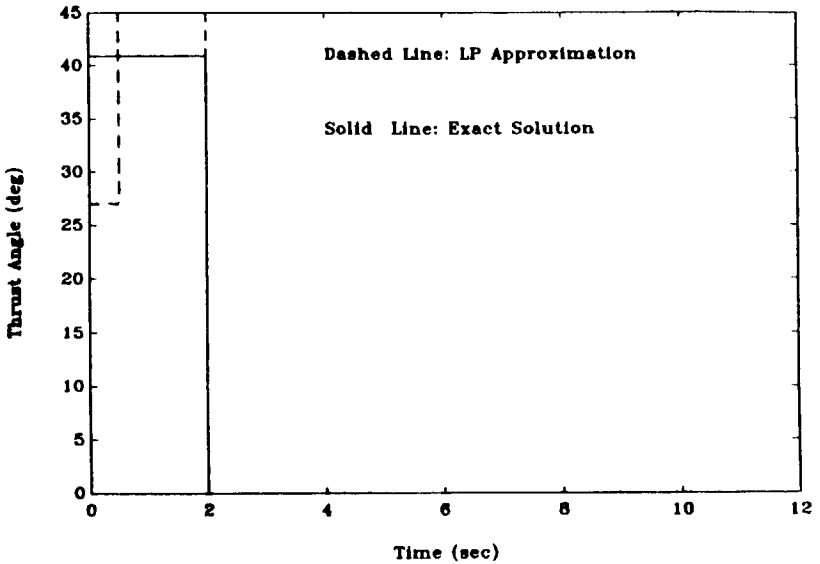

Fig. 3 Thrust Angle Time History for Ground-to-Ground Missile Range Maximization

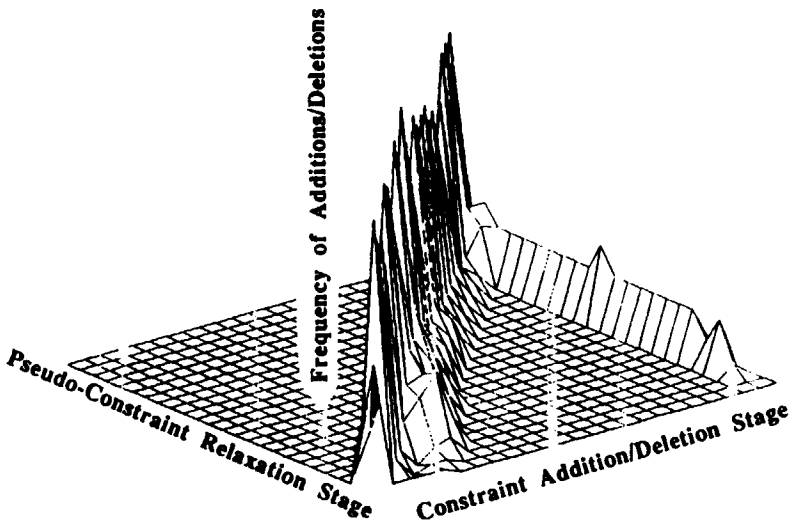

Fig. 4 Two-Dimensional Histogram of Frequency of Constraint Additions and Deletions at a Given Stage for a Given Pseudo-Constraint Relaxation Stage